Technical Disclosure Commons

Defensive Publications Series

March 2020

# Adaptive Remote Execution

Victor Carbune

Alexandru Damian

# Adaptive Remote Execution

**Abstract:**

This publication describes systems and techniques for adaptive remote execution of computing tasks from a mobile computing device, such as a smartphone. The computing resources required by applications (and/or computation blocks thereof) can vary significantly due to workload and other factors. Execution of resource-intensive computation blocks can quickly drain battery power and overwhelm limited on-device resources, resulting in poor user experience. In some circumstances, offloading compute tasks through low-latency, high-speed networks can be advantageous. The described systems and techniques determine optimal offload decisions for respective compute blocks; the decisions may be adapted to current conditions (*e.g.,* network connectivity, network reliability, device load, and/or the like), characteristics of the respective compute blocks (*e.g.,* battery consumption, execution time, and/or the like), and so on. Offload decisions may be determined in accordance with a machine-learned modeling approach based on an artificial neural network, Contextual Bandits, Reinforcement Learning, and/or the like.

**Keywords:**

Electronic device, mobile computing device, smartphone, tablet, laptop, smartwatch, operating system, battery level, battery usage, battery life, network connectivity, network reliability, remote execution, machine learning, machine-learned model, artificial neural network, Contextual Bandits, Reinforcement Learning.

**Background:**

The computing resources required to execute applications on a mobile device can vary significantly depending on workload and/or other factors. On-device execution of computationally intensive applications can quickly drain energy reserves and, if available computing resources are insufficient, can result in a high-latency, slow response time, and a poor user experience. Remote execution over reliable, high-speed, low-latency networks can produce significant benefits, such as reduced battery consumption, faster response time, and better overall user experience. However, these benefits may only be achieved under certain conditions, *e.g.,* when offloading certain types of compute blocks and, even then, only when network connectivity is sufficiently fast and reliable. Although techniques for remote execution are widely available, it can be difficult to determine optimal offload decisions employing these techniques that account for dynamically changing conditions. Therefore, it is desirable to provide an adaptive model to determine whether to offload particular compute blocks for remote execution to produce optimal results.

**Description:**

As illustrated in FIG. 1 below, an electronic device 100, such as a smartphone, can include an offload manager 110 to determine optimal offload decisions for execution of respective compute blocks 112. A compute block 112 can include an application and/or a portion thereof (a method, function, subroutine, or the like). In response to a request to execute a compute block 112, the offload manager 110 can determine whether to: a) offload execution of the compute block 112, or b) execute the compute block 112 by use of local resources 120 (*e.g.,* on-device processor, memory, and/or the like). An offloaded compute block 112 can be executed by use of network-accessible computing resources (remote compute nodes 130) through the network 102 (*e.g.,* by

remote method invocation (RMI), remote procedure calls (RPC), Simple Object Access Protocol (SOAP), and/or other suitable techniques).

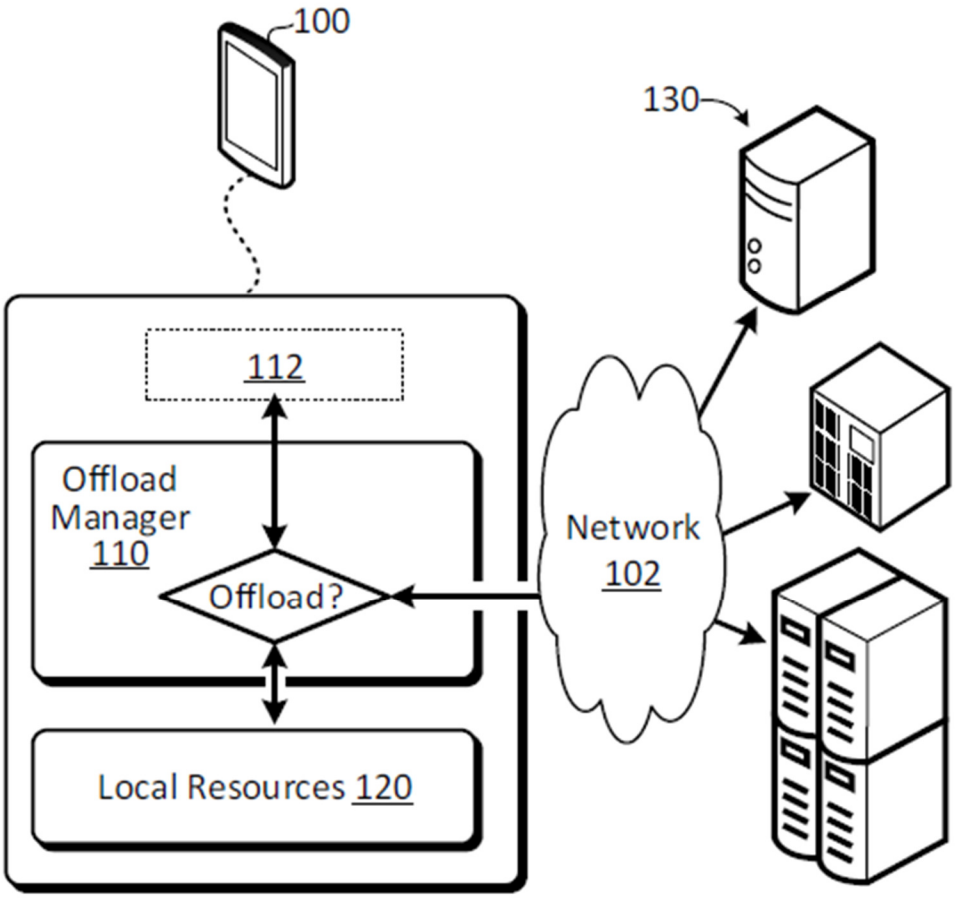


**FIG. 1 Electronic device with Execution Manager**

As illustrated in FIG. 2, the offload manager 110 can determine optimal offload decisions 212 for compute blocks 112 based on a current state of the electronic device 100, network 102, characteristics of the compute blocks 112, feedback from previous offload decisions 212, and so on.

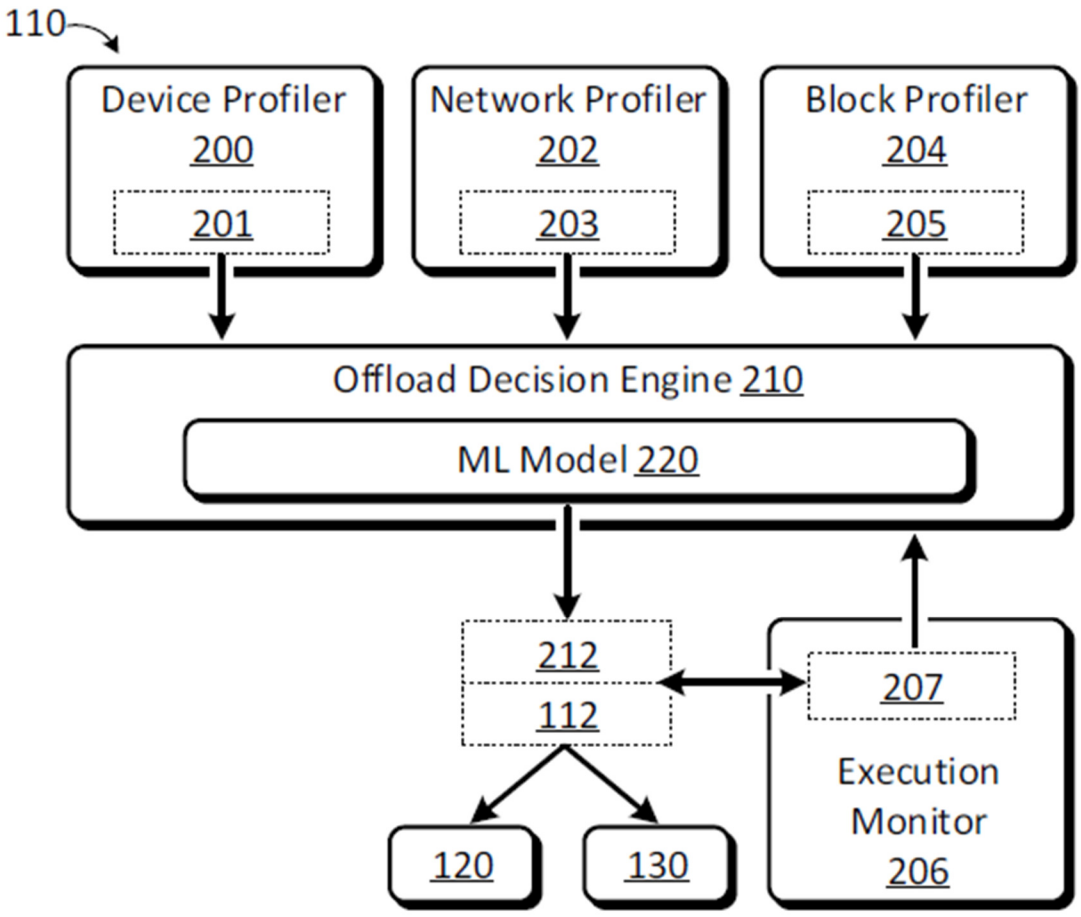


**FIG. 2 Offload Manager Operating on an Electronic Device**

A device profiler 200 determines device features 201 that are adapted to characterize the current state of the electronic device 100 (and/or local resources 120 thereof), such as available energy reserves (*e.g.,* remaining battery capacity), processor capabilities, current processor load, memory capacity, current memory availability, and so on. A network profiler 202 determines network features 203 that are adapted to characterize the current state of the connection of the electronic device 100 to the network 102 (and/or respective remote compute nodes 130), such as connection speed, bandwidth, latency, timeouts, packet loss, and/or the like. A block profiler 204 determines block features 205 that are adapted to characterize compute blocks 112 being evaluated for remote execution, such as machine code of the compute blocks 112, workload (*e.g.,* input parameters of the compute blocks 112), typical battery consumption, typical latency, typical

response time, execution history (*e.g.,* whether the compute blocks 112 were offloaded in the past along with observed gains), and so on. Characteristics of respective compute blocks 112 (*e.g.,* block features 205) can be maintained in non-transitory storage of the electronic device 100 or included and/or stored with the respective compute blocks 112. An offload decision engine 210 determines offload decisions 212 based on, *inter alia*, the described device features 201, network features 203, and/or block features 205. In some aspects, the offload decisions 212 quantify the degree to which current network conditions are suitable for offloading (*e.g.,* a value between 0 and 1, where 0 indicates unsuitable network conditions and 1 indicates ideal network conditions). As illustrated, the offload decision 212 determined for a compute block 112 may indicate whether to: a) offload execution of the compute block 112 to a remote compute node 130, or b) execute the compute block 112 by use of local resources 120 of the electronic device 100.

An execution monitor 206 can determine feedback 207 for respective offload decisions 212. The feedback 207 determined for an offload decision 212 may characterize results of the offload decision 212 in terms of*, inter alia*, response time or latency (time required to execute the compute block 112 either locally or by the selected remote compute node 130), battery drain, local resource consumption, bandwidth consumption, and/or the like. The feedback 207 determined for respective offload decisions 212 may be incorporated into block features 205 of the corresponding compute blocks 112.

In some aspects, the offload decisions 212 are determined by a machine-learned (ML) model 220, such as an artificial neural network (ANN) that takes as input the features described above (device features 201, network features 203, and/or block features 205) and outputs offload decisions 212 that: a) quantify the degree to which current network conditions are suitable for remote execution, and/or b) designate whether execution of respective compute blocks 112 should

be offloaded. The ANN can be trained to produce optimal offload decisions 212 by use of, *inter alia*, feedback 207 and/or training data (*e.g.,* previous decisions 212 determined to result in optimal feedback 207).

As illustrated in FIG. 3, the ML model 220 of the offload decision engine 210 can implement a Reinforcement Learning (RL) algorithm or Contextual Bandits. In FIG. 3, the environment represents current operating conditions of the electronic device 100, network 102, compute block(s) 112 to be executed thereby, and so on. The state 301 of the environment can be acquired based on the features described herein (*e.g.,* device features 201, network features 203, block features 205, and/or the like). The ML model 220 determines actions 302 based on the current state 301 of the environment; the actions 302 determined by the ML model 220 may comprise network suitability and/or offload decisions 212 for respective compute blocks 112. The reward 307 used to train (and/or reinforce the training) of the ML model 220 can be derived from the feedback 207 determined for respective actions 302 (offload decisions 212) in accordance with a suitable reinforcement learning technique, such as Markov Decision Process (MDP), Q learning, or the like.

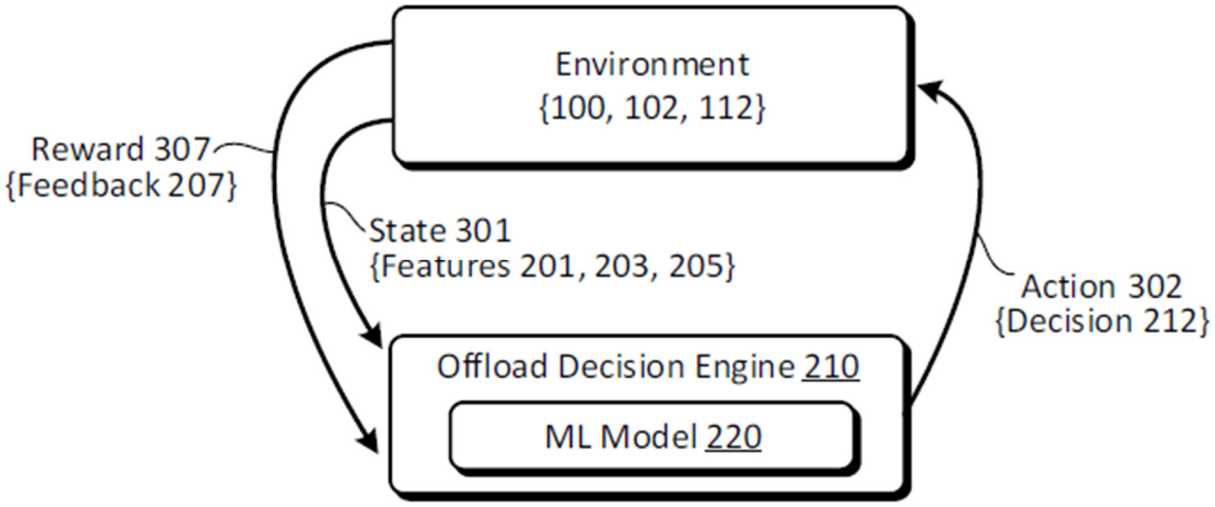


**FIG. 3 Contextual Bandit Learning Model of Offload Decision Engine**

As illustrated in FIG. 4, in some aspects, the ML model 220 of the offload decision engine 210 implements an RL architecture where actions 302 affect state 301, and rewards 307 may be delayed in time.

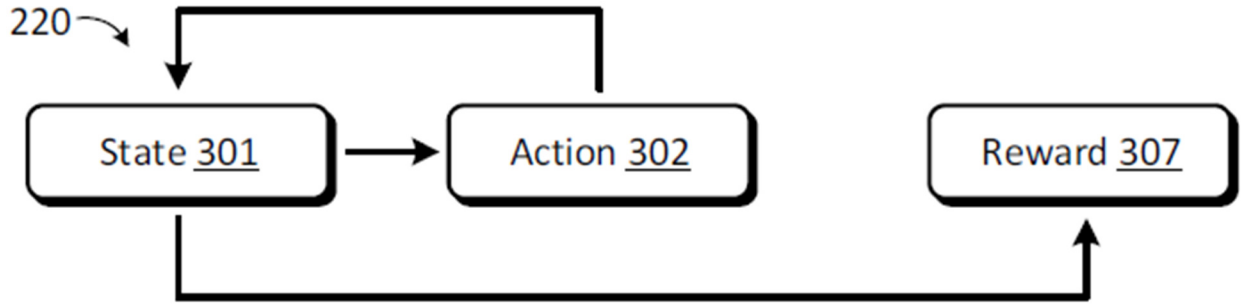


**FIG. 4 Reinforcement Learning Model of Offload Decision Engine**

As above, the state 301 can be acquired by the offload decision engine 210 (by the device profiler 200, network profiler 202, block profiler 204, and/or execution monitor 206). The state 301 (*e.g.,* device features 201, network features 203, block features 205) may be received at an input layer of the ML model 220, which in response, produces corresponding actions 302 (offload decisions 212). As illustrated above, the actions 302 may affect the state 301 by which offload decisions 212 for subsequent compute blocks 121 are determined before corresponding rewards

307 are received; *e.g.,* consume local resources 120, network bandwidth, and/or the like during execution (and before completion), which can enable the ML model 220 of the offload decision engine 210 to model compute block dependencies (*e.g.,* compute blocks 112 that depend on outputs produced by execution of other compute blocks 112).

**References:**

[1] International Patent Publication No. WO 2017067586 A1, "Method and system for code offloading in mobile computing," priority to October 21, 2015.

[2] International Patent Publication No. WO 2019209154 A1, "Mechanism for machine learning in distributed computing," priority to April 27, 2018.